From linear models to deep neural networks

Deep learning course for industry

Mitko Veta

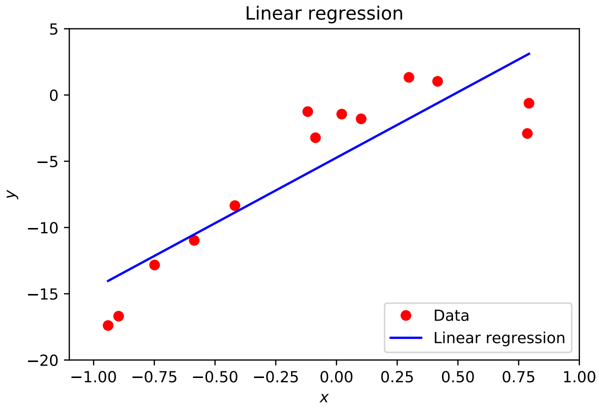
Eindhoven University of Technology
Department of Biomedical Engineering

2020

Learning goals

- Introduce a linear model for classification.
- Demonstrate how linear classification models can be combined to produce more complex decision boundaries.
- Introduce the layered view of neural networks.

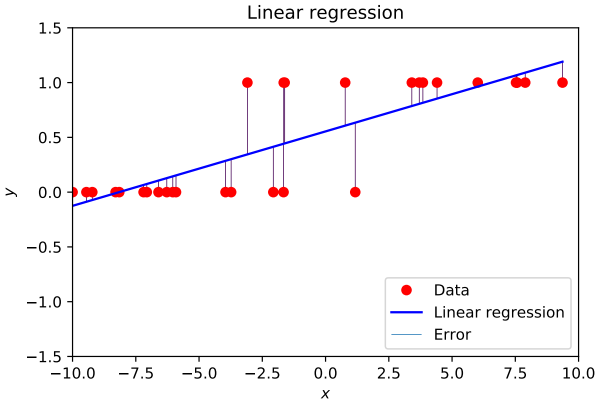
Previously: linear model for regression



$$\hat{y} = \hat{w}_0 + \sum_{j=1}^{p} x_j \hat{w}_j$$

Question: Can this model be used for classification (e.g. binary classification)?

Linear regression for a binary target



$$\hat{y} = \hat{w}_0 + \sum_{j=1}^{p} x_j \hat{w}_j$$

Towards a linear model for classification

$$\hat{y} = \hat{w}_0 + \sum_{j=1}^{p} x_j \hat{w}_j$$

The following changes need to be made to the linear model:

- Instead of directly predicting the value $\hat{y}$ (which is not continuous value), predict the probability that the sample belongs to one of the classes, e.g. $p(y_i = 1|\mathbf{x}_i)$.

Towards a linear model for classification

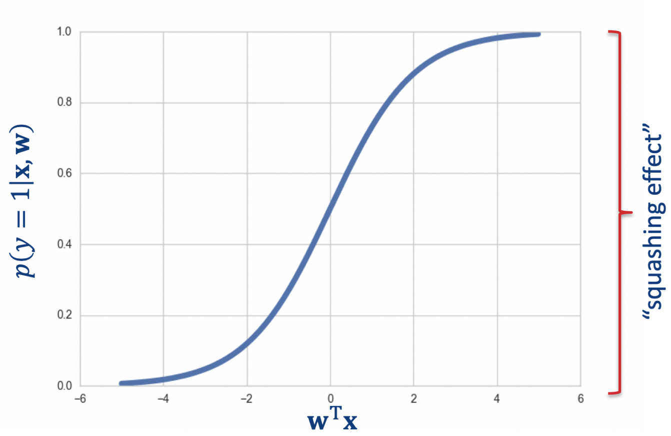
$$\hat{y} = \hat{w}_0 + \sum_{j=1}^{p} x_j \hat{w}_j$$

The following changes need to be made to the linear model:

- Instead of directly predicting the value $\hat{y}$ (which is not continuous value), predict the probability that the sample belongs to one of the classes, e.g. $p(y_i = 1|\mathbf{x}_i)$.
- $p(y_i = 1|\mathbf{x}_i)$ is a continuous value, however, it is bounded between 0 an 1.
- In order to interpret is as a probability, $\hat{w}_0 + \sum_{j=1}^{p} x_j \hat{w}_j$ has to be "squashed" between 0 and 1.

The sigmoid function

$$\sigma(a) = \frac{1}{1+e^{-a}}$$



Logistic regression: a linear model for classification

Linear regression: $\hat{y} = \hat{w}_0 + \sum_{j=1}^{p} x_j \hat{w}_j$

Logistic regression: $p(y_i = 1 | \mathbf{x}_i) = \sigma(\hat{w}_0 + \sum_{j=1}^{p} x_j \hat{w}_j)$

Logistic regression: another view

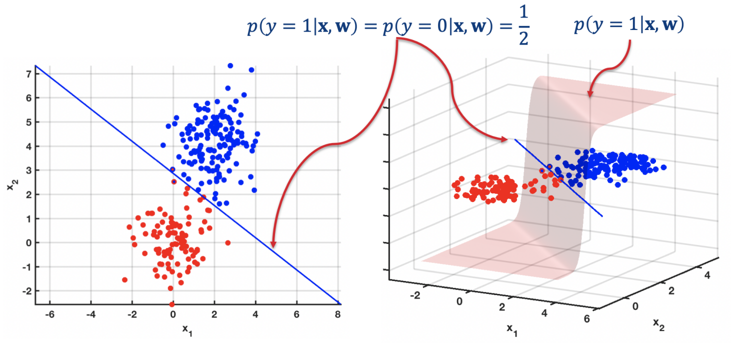
The log-odds that the sample belongs to class "1" are modelled with a linear model:

$$log(\frac{p(y_i=1|\mathbf{x}_i)}{p(y_i=0|\mathbf{x}_i)}) = \hat{w}_0 + \sum_{j=1}^{p} x_j \hat{w}_j$$

$$log(\frac{p(y_i=1|\mathbf{x}_i)}{1-p(y_i=1|\mathbf{x}_i)}) = \hat{w}_0 + \sum_{j=1}^{p} x_j \hat{w}_j$$

Logistic regression: a linear model for classification

Logistic regression produces a linear decision boundary:



Fitting logistic regression

Given a training dataset $\{\mathbf{x}_i, y_i\}$, the parameters $\mathbf{w}$ of the logistic regression model can be estimated by minimising the negative log-likelihood (NLL) of the prediction $p(y_i = 1|\mathbf{x}_i)$:

$$J(\mathbf{w}) = -\sum_{i}^{N} \log \left[p(y_i = 1|\mathbf{x}_i, \mathbf{w})^{y_i} p(y_i = 0|\mathbf{x}_i, \mathbf{w})^{1-y_i} \right]$$

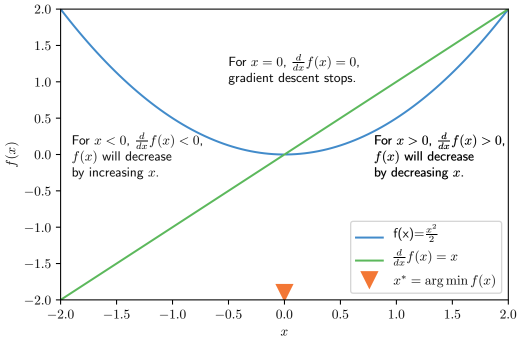
$$\hat{\mathbf{w}} = \underset{\mathbf{w}}{\operatorname{argmin}}\, J(\mathbf{w})$$

This is, in fact, equivalent to minimising the cross-entropy between the predictions $p(y_i = 1|\mathbf{x}_i)$ and the ground truth y_i.

Gradient descent

Compared to linear regression and the residual sum of squares function, in the case of logistic regression there is no closed-form solution for the parameters that minimise the NLL.

The optimal parameters are found with a numerical procedure called gradient descent.



Gradient descent

The gradient descent algorithm:

- Initialise the parameters $\mathbf{w}$ to some random values.
- While some stopping criterion is not met (e.g. maximum number of iterations):
    - Compute the gradient $\nabla_{\mathbf{w}} J(\mathbf{w})$.
    - Update the current estimate of the parameters in the direction opposite of the gradient (in order to move towards the minimum of the function): $\mathbf{w} \leftarrow \mathbf{w} - \mu \nabla_{\mathbf{w}} J(\mathbf{w})$

Gradient descent

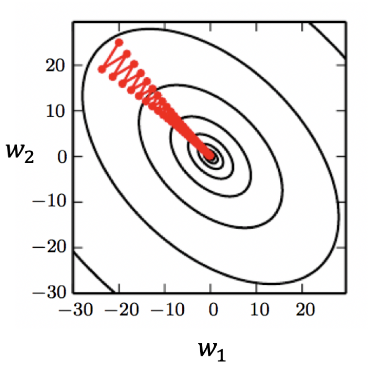
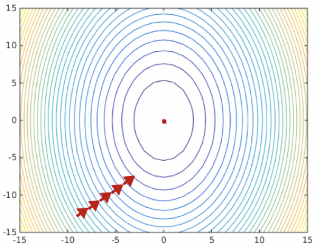


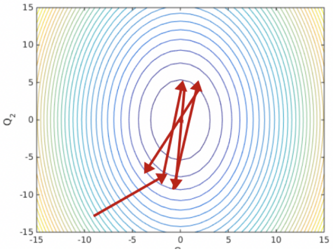
Figure source: deeplearningbook.org

Learning rate

The choice of the learning rate μ is crucial.



Too low.



Too high.

Stochastic gradient descent

Note that in order to compute $\nabla_{\mathbf{w}} J(\mathbf{w})$, the output of the model for all N training samples needs to be computed.

This is computationally challenging in case of large number of training samples.

Solution: estimate $\nabla_{\mathbf{w}} J(\mathbf{w})$ based on a smaller number of training samples $N' << N$.

Stochastic gradient descent

The algorithm is called stochastic gradient descent (SGD).

Each iteration of the gradient descent algorithm, a random subset of N' training samples is sampled from the entire training set.

N' is called the **batch size**.

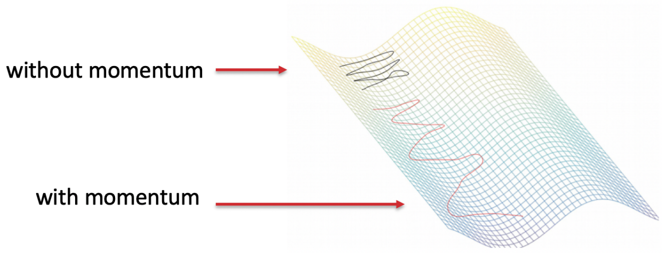
Stochastic gradient descent with momentum

A common variant of SGD is SGD with momentum.

Intuition: add speed in the average direction. This variant often converges a lot faster than regular SGD.

$$\nu \leftarrow \alpha\nu - \mu\nabla_{\mathbf{w}} J(\mathbf{w})$$

$$\mathbf{w} \leftarrow \mathbf{w} + \nu$$



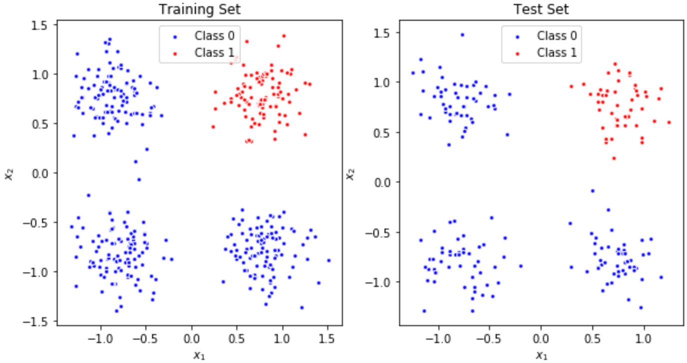
Softmax regression

The extension of logistic regression to multi-class problem (more than two classes) is straightforward and it is called softmax regression.

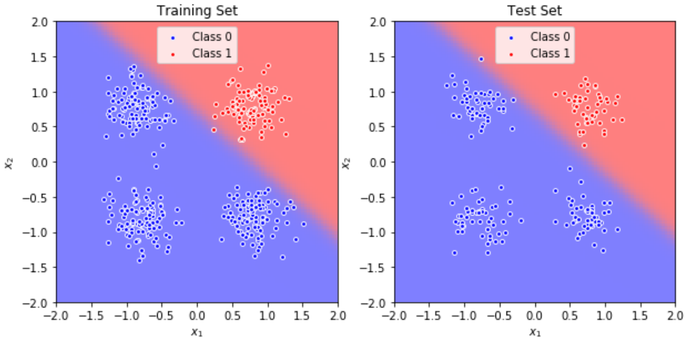
Essentially, for C number of classes, C number of linear models are and are converted to probabilities using the softmax function:

$$p(y = i|\mathbf{x}) = \frac{e^{\mathbf{x}^\intercal \mathbf{w}_i}}{\sum_k^C e^{\mathbf{x}^\intercal \mathbf{w}_k}}$$

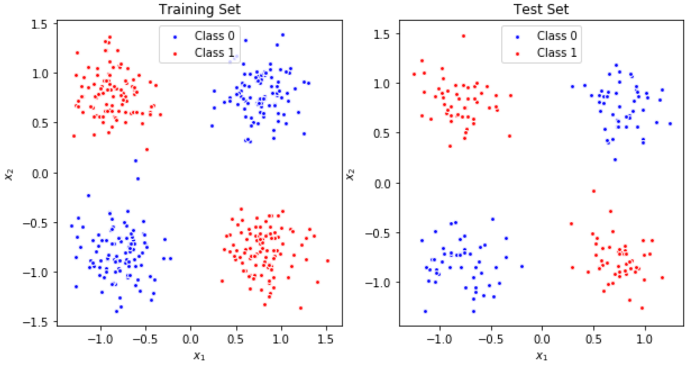
The AND classification problem



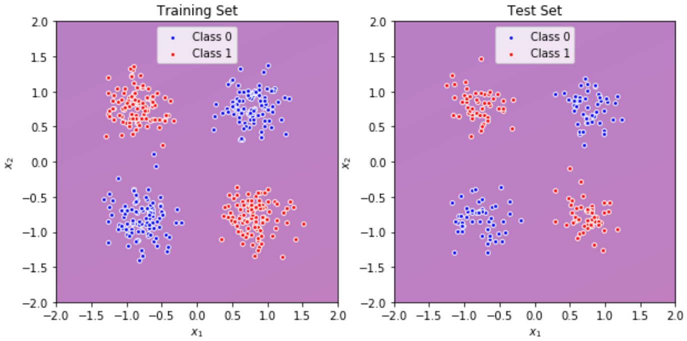
The AND classification problem: logistic regression



The XOR classification problem

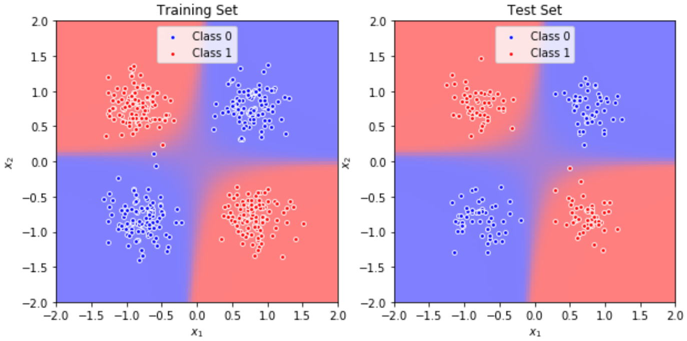


The XOR classification problem: logistic regression



The XOR classification problem: LR + polynomial transformation

$$\mathbf{x}_i = [x_1, x_2] \rightarrow \tilde{\mathbf{x}}_i = [x_1, x_2, x_1^2, x_2^2, x_1 x_2]$$



Feature transformation

Prior to 2010, most of the work on machine learning involved *relatively* simple classifiers in combination with extensive, “manual” feature engineering.

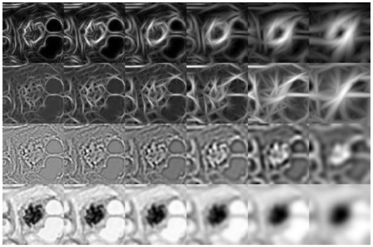
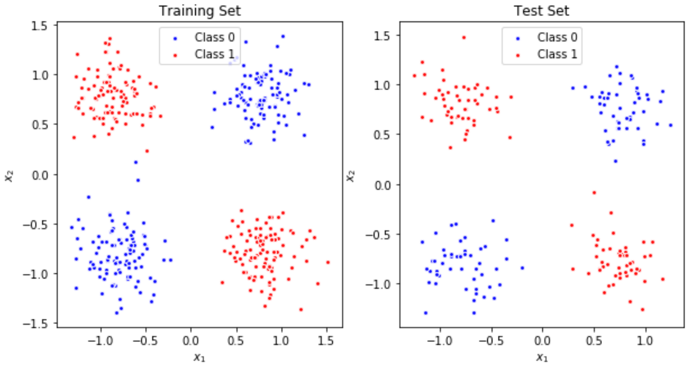


Figure source: Veta et al. SPIE MI 2012

A change of paradigm

Feed **raw** (or minimally processed) images directly to the machine learning models. Design the models in such a way that they can **learn the needed feature transformations** directly from the image data.

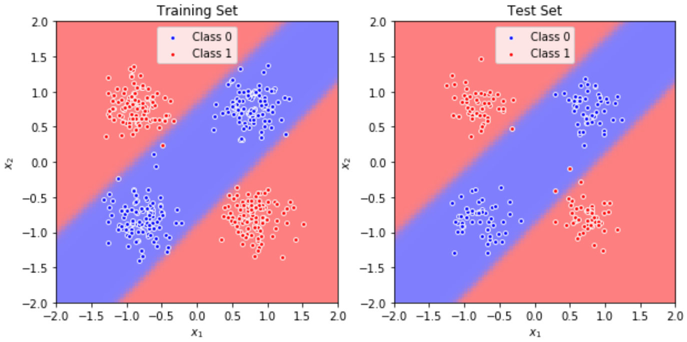
The XOR classification problem



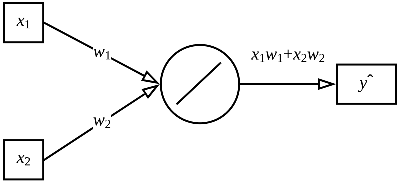
Question: How can the logistic regression classifier be modified to solve the XOR problem without explicit feature transformation?

Combining two linear classifiers

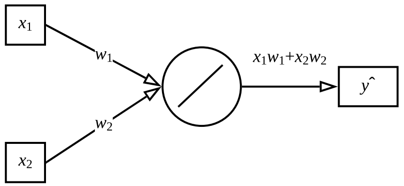
Answer: Use two logistic regression classifiers.

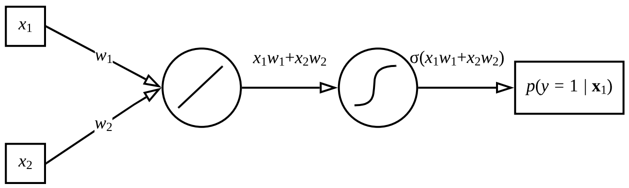


A graphical view of linear and logistic regression

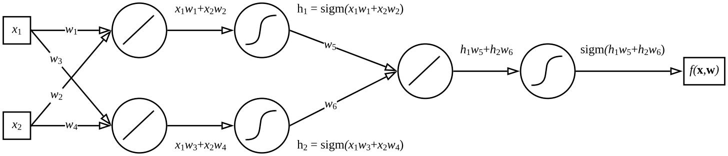


A graphical view of linear and logistic regression



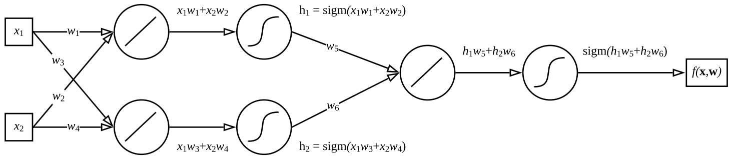


Combining two linear classifiers



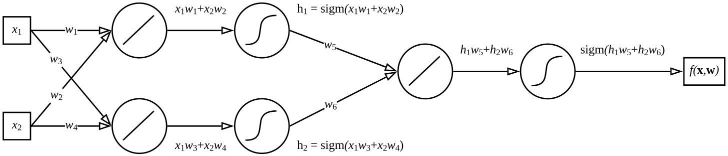
x_1
x_2
w_1
w_2
w_3
w_4
$x_1w_1+x_2w_2$
$x_1w_3+x_2w_4$
$h_1 = \text{sigm}(x_1w_1+x_2w_2)$
$h_2 = \text{sigm}(x_1w_3+x_2w_4)$
w_5
w_6
$h_1w_5+h_2w_6$
$\text{sigm}(h_1w_5+h_2w_6)$
$f(\mathbf{x},\mathbf{w})$

Combining two linear classifiers


x_1
x_2
w_1
w_2
w_3
w_4
$x_1w_1+x_2w_2$
$x_1w_3+x_2w_4$
$h_1 = \mathrm{sigm}(x_1w_1+x_2w_2)$
$h_2 = \mathrm{sigm}(x_1w_3+x_2w_4)$
w_5
w_6
$h_1w_5+h_2w_6$
$\mathrm{sigm}(h_1w_5+h_2w_6)$
$f(\mathbf{x},\mathbf{w})$

This is a (small) feedforward neural network.

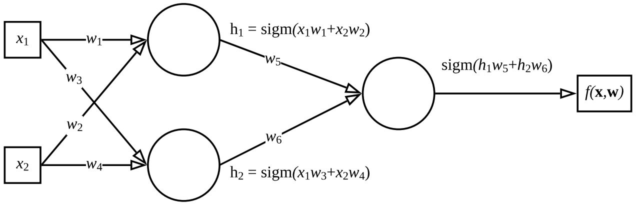
Combining two linear classifiers



This is a (small) feedforward neural network.

It is a composition of two functions $\mathbf{h}(\mathbf{x})$ and $f(\mathbf{x})$. $\mathbf{h}(\mathbf{x})$ is called a hidden layer. It can be seen as a **learned** feature representation of $\mathbf{x}$ (analogous to the hand-crafted features $\tilde{\mathbf{x}}$).

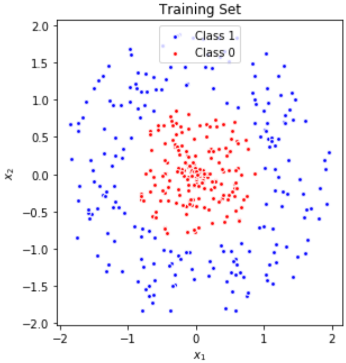
Combining two linear classifiers

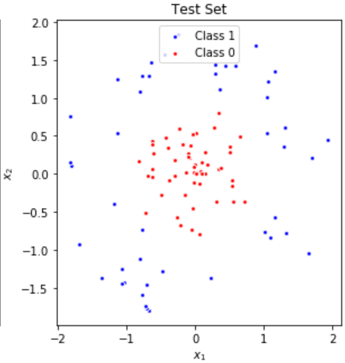


x_1
x_2
w_1
w_2
w_3
w_4
w_5
w_6
$h_1 = \mathrm{sigm}(x_1 w_1 + x_2 w_2)$
$h_2 = \mathrm{sigm}(x_1 w_3 + x_2 w_4)$
$\mathrm{sigm}(h_1 w_5 + h_2 w_6)$
$f(\mathbf{x},\mathbf{w})$

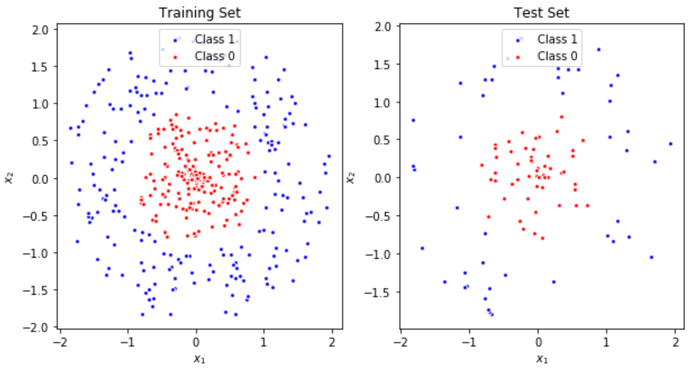
Often the sigmoid nonlinarity is not depicted in graphical representations (but it is there, and as shown later, it is crucial).

A somewhat more difficult problem





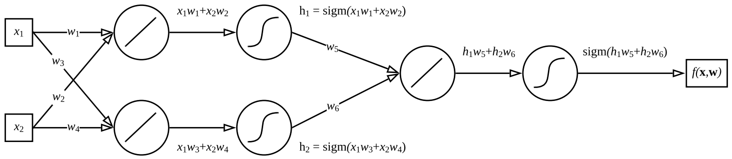
A somewhat more difficult problem



Example in Tensorflow Playground.

Why do we need nonlinearities

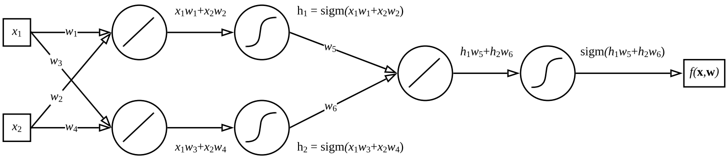
Question: What will happen if we omit the sigmoid nonlinearities in the hidden layer?



x_1
x_2
w_1
w_3
w_2
w_4
$x_1w_1+x_2w_2$
$x_1w_3+x_2w_4$
$h_1 = \mathrm{sigm}(x_1w_1+x_2w_2)$
$h_2 = \mathrm{sigm}(x_1w_3+x_2w_4)$
w_5
w_6
$h_1w_5+h_2w_6$
$\mathrm{sigm}(h_1w_5+h_2w_6)$
$f(\mathbf{x},\mathbf{w})$

Why do we need nonlinearities

Question: What will happen if we omit the sigmoid nonlinearities in the hidden layer?

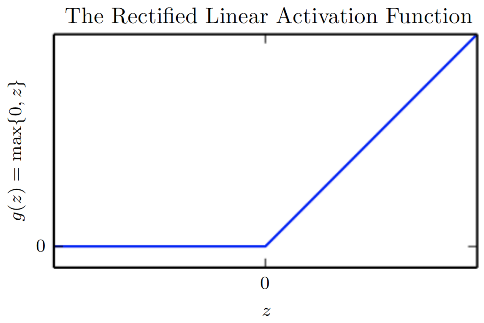


x_1
x_2
w_1
w_2
w_3
w_4
$x_1w_1+x_2w_2$
$x_1w_3+x_2w_4$
$\mathrm{h}_1 = \mathrm{sigm}(x_1w_1+x_2w_2)$
$\mathrm{h}_2 = \mathrm{sigm}(x_1w_3+x_2w_4)$
w_5
w_6
$h_1w_5+h_2w_6$
$\mathrm{sigm}(h_1w_5+h_2w_6)$
$f(\mathbf{x},\mathbf{w})$

Without the nonlinearities, the network will be a linear combination of linear combinations of the input features, which collapses to a linear combination of the input features.

In other words, it will be no different than just logistic regression. **The network has no depth.**

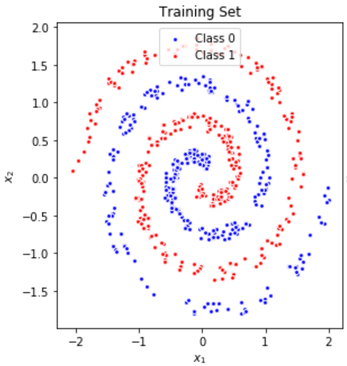
The ReLU nonlinearity

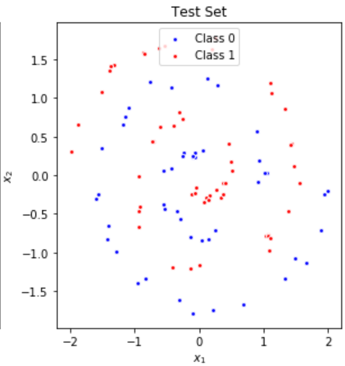


Instead of a sigmoid, the recommended default nonlinearity in modern neural network is the rectified linear unit. Because rectified linear units are nearly linear, they preserve many of the properties that make linear models easy to optimise with gradient-based methods.

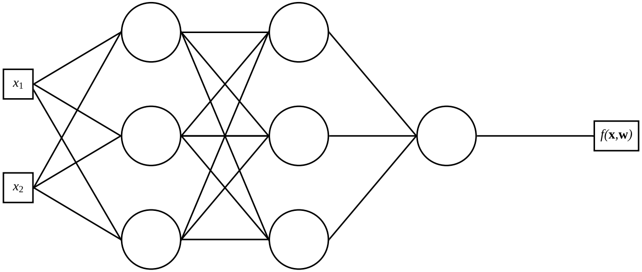
Figure from: deeplearningbook.org

An even more difficult problem

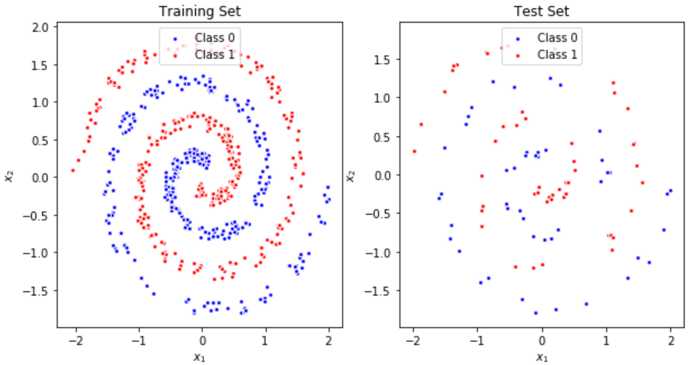




Multilayer neural networks

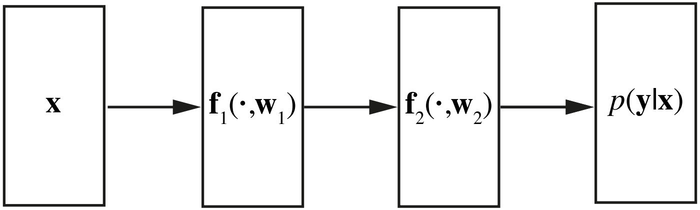


An even more difficult problem



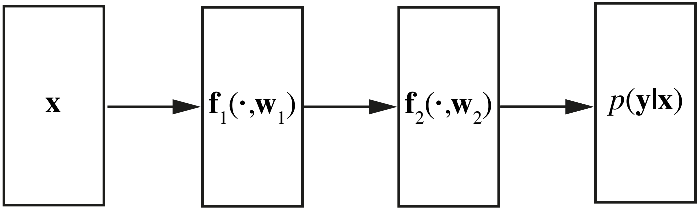
Example in Tensorflow Playground.

The layered view of neural networks


$\mathbf{x}$
$\mathbf{f}_1(\cdot,\mathbf{w}_1)$
$\mathbf{f}_2(\cdot,\mathbf{w}_2)$
$p(\mathbf{y}|\mathbf{x})$

The neural network can be seen as a composition of the different layers: $f_d(\ldots f_2(f_1(\mathbf{x})))$. In modern deep learning frameworks layers (not neurons) are considered the basic building blocks.

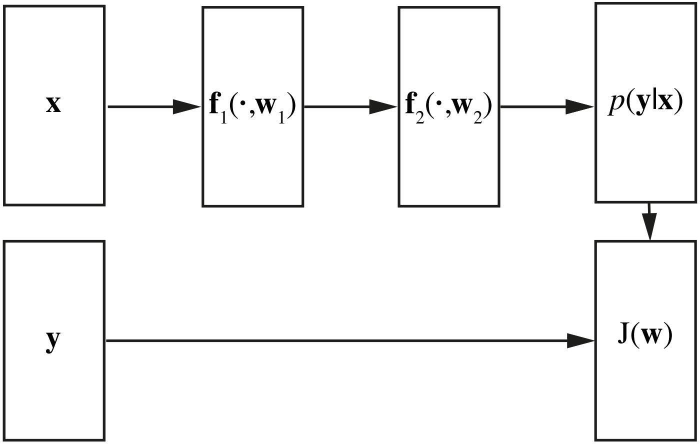
The layered view of neural networks


$\mathbf{x}$
$\mathbf{f}_1(\cdot,\mathbf{w}_1)$
$\mathbf{f}_2(\cdot,\mathbf{w}_2)$
$p(\mathbf{y}|\mathbf{x})$

The neural network can be seen as a composition of the different layers: $f_d(\dots f_2(f_1(\mathbf{x})))$. In modern deep learning frameworks layers (not neurons) are considered the basic building blocks.

The number of layers "d" is called the **depth** of the network. This is where the "deep" on deep learning comes from.

The layered view of neural networks



$\mathbf{x}$
$\mathbf{f}_1(\cdot,\mathbf{w}_1)$
$\mathbf{f}_2(\cdot,\mathbf{w}_2)$
$p(\mathbf{y}|\mathbf{x})$
$\mathbf{y}$
$J(\mathbf{w})$

Layered view of a neural network at training time.

Summary

- The linear regression model can be extended to a logistic regression model for classification by appending a sigmoid nonlinearity (the model then is set to predicts the probability of a class membership).
- Simple linear classifiers such as logistic regression can be combined in neural networks that can solve more complex problems.
- In modern deep learning frameworks layers (not In modern deep learning frameworks layers (not neurons) are considered the basic building blocks.